Web resource characterization, detection and counter-measures

immediate

Abstract

1 Preliminary steps

1.1 Crawling

- Test OpenWPM
- Test referrer chain extraction from OpenWPM: if not possible
  - Test iframe extraction with OpenWPM/headless Chrome

1.2 Malicious domain/URL detection

State of the art:

- [4]:
  - JS/iframe analysis
  - known malware distribution site
  - VM-based verification

  ⇒ too much work for us (we are not Google :))
- [3]:
  - Safe-browsing API
  - Microsoft Forefront 2010 ⇒ now discontinued
- [5]:
  - Wepawet ⇒ now discontinued
  - Blacklist consensus (5 in 49 blacklists [1], and later [2]))
  - Send to Virstotal ⇒ maybe a bit dangerous

  Main idea: Safe-browsing and/or blacklist
- Check Safe browsing API
- Check blocking lists used in [5]: [1, 2]

2 Steps

- Crawl
  - ☑ OpenWPM/headless Chrome
  - ☑ Rebuild resource chains (referer, HTML 302 and JS)
  - ☑ Malicious resource chains as chains of URL with malicious domain ; same as [4, 3] but not only for ads
- Compare our results with existing ones
  - ☐ Rebuild same statistics
    - ☐ Crawling statistics: Table 1 of [3]
  - ☐ Remake figures from other papers (Note: I think ad networks [5] has a meaning similar to node [3] although ad networks may also represents the second level domain name of nodes which would mean that some ad networks may contain several nodes) ; suggestions are kind of sorted by relevance
    - ☐ Ad networks involved in ad arbitration for malicious and benign advertisements: Figure 5 of [5]
    - ☐ Malvertising distribution from selected ad networks: Figure 1 of [5]
    - ☐ Global distribution from selected ad networks: Figure 2 of [5]
    - ☐ Safe Browsings delays in detection: Figure 12 of [3]
    - ☑ Websites categorization that served malvertisements: Figure 3 of [5] Figure 4 of [4] **[Comment:** *is it about malicious URL/ressource in chain or publisher?* – Johan]
    - ☑ Malvertisement distribution based on top level domains: Figure 4 of [5] – DONE FOR ALL RESOURCES WITHOUT BREAKDOWN
    - ☐ Node-pair frequency: Figure 6 of [3]
    - ☐ For each node on the malvertising paths, the minimum distance to a node detected by Safebrowsing or Forefront (alarm node) vs. the number of infected publishers that it is associated with: Figure 7 of [3]
    - ☐ CDF of the durations between the registration dates and the expiration dates of Web domains: Figure 5 of [3] – IN PROGRESS
- Malicious ressources analysis:
  - ☑ Breakdown by country: Table 2 of [3] **[Comment:** *is it about malicious URL/ressource in chain or publisher?* – Johan]
  - ☐ Analyze how ads-related and non-ad-related malicious ressource are present in the Alexa Top 100k (rationale: we do not see many ad-related malicious ressources and we see many malicious ressources directly embedded on publisher/first party, are these ads actually present on publisher/first party that are higher in Alexa Top than publisher/first party that directly embedding malicious ressources?)
    - ☐ ECDF/distribution of # of malicious ressources linked to ads and non-ads with the position of the first party in the Alexa Top 100k on x axis:
  - ☐ Build referer chain general statistics:
    - ☐ Length
    - ☑ DNS registration as in [3] – **[Comment:** *is it about malicious URL/ressource found in chain?* – Johan]
- Evaluate counter-measures:

  - Check if blocked by ad- and privacy-focused blocking list/browser extensions (blocking list suggestions here: `https://github.com/gorhill/uBlock/wiki/Blocking-mode`)
    * Match against public list using `https://github.com/scrapinghub/adblockparser` ; what about closed-source ones (Ghostery) ?
    * Rerun in OpenWPM
  - Analysis
    ☐ Characteristics of blocked and not-blocked referer chains
      ☐ Length
      ☐ ???
    ☐ Position of blocked element in referer chain
      ☐ Distance to publisher
      ☐ Distance to URL/ressource flagged by SafeBrowsing

- Analyze and detect
  ☐ Characterize current malicious chains properties (compare with [3])
  ☐ Apply ML as in [3]
    ☐ Check if faster than Safe browsing
    ☐ New ML techniques (beyond decision tree + pruning)

3 Resource chain extraction

3.1 Terminology

Delivery chain [4] and ad path [3] are both the same things. Delivery chain sounds really security-related while ad path seems too advertising-related. That is why I changed the name to resource chain. But suggestion are welcome. :)

3.2 Concerned sections in papers

- [4]: Constructing the Malware Distribution Networks subsection
- [3]: section 3.1

3.3 Redirection identification techniques

- HTML code [4, 3] and HTTP 302 [3]:
  - Referer is used [4, 3]
  - "Location field of a requests response (Request As response redirects the browser to URL B)" [3]
- Javascript [4, 3]: cannot link URL request to JS script because referer is the visited website (top_level_url in OpenWPM) not the script's URL

3.4 How existing work detect JS redirection ?

3.4.1 [4]

- Extract and "interpret" (run ?) JS code
- Extract URL from JS code
- Match requested URL to URL in JS code using edit distance

3.4.2 [3]

- Extract JS code
- Extract URL from JS code
- Search for complete match between requested URL to URL in JS code
- If no complete match (e.g. in [3] : the JS code concatenate several strings to build a URL), try to match domain name of requested URL and JS script

3.5 Suggested techniques for referer extraction in the context of OpenWPM

3.5.1 HTML-related resource chains building

- HTML code: use referer and url column in http_requests table as in [4, 3]
- HTML response with 302: if a HTTP request (http_requests table) located after an HTTP response (http_responses table) whose status column is 302 has a url column equal to the location column of this HTTP response, then the referer of this HTTP request is the url column in the HTTP response [3]

3.5.2 JS-related resource chains building

- Collect HTML code (with JS inside) using CommandSequence::recursive_dump_page_source
- Extract URLs inside the JS code
- Extract domain of JS scripts
- If an URL request has the top_level_url as referer: try to match a URL request with extracted URLs in the JS code (as in [4, 3]) *and* the JS scripts' domains (as in [3]) using the edit distance (as in [4]), URL with smallest distance wins

3.5.3 Detailed procedure

- Collect data: http_requests and http_response tables, and HTML code (with JS inside) using CommandSequence::recursive_dump_page_source
- Extract 302-related referer links: if a HTTP request (http_requests table) located after an HTTP response (http_responses table) whose status column is 302 has a url column equal to the location column of this HTTP response, then the referer of this HTTP request is the url column in the HTTP response [3]
- If referer in http_requests is not equal to top_level_url and not empty:
  - Use referer in http_requests as referer

  else
  - Extract JS script domains
  - Extract URLs inside the JS code
  - Process the edit distances between a URL request and extracted URLs in the JS code (as in [4, 3]) *and* the JS scripts' domains (as in [3])
  - If all distances are bigger than the size diference between the URL length and the smallest JS script domain length, i.e. URL does not completely contain any JS script domains and also does not contain other URL found inside the JS scripts (they would have been bigger and thus yielded a smaller edit distance)

* top_level_url is considered as the referer

else

* URL found among JS script domain and URL found in JS scripts with smallest distance becomes the referer

References

[1] Marc Kührer and Thorsten Holz. An empirical analysis of malware blacklists. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, 35(1):11–16, 2012.

[2] Marc Kührer, Christian Rossow, and Thorsten Holz. Paint it black: Evaluating the effectiveness of malware blacklists. In *International Workshop on Recent Advances in Intrusion Detection*, pages 1–21. Springer, 2014.

[3] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 674–686. ACM, 2012.

[4] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Monrose Fabian. All your iframes point to us. 2008.

[5] Apostolis Zarras, Alexandros Kapravelos, Gianluca Stringhini, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 373–380. ACM, 2014.